

A Performance Model for the Communication in Fast Multipole Methods on HPC Platforms

Huda Ibeid, Rio Yokota, and David Keyes

Division of Computer, Electrical and Mathematical Sciences and Engineering
King Abdullah University of Science and Technology, Saudi Arabia

Abstract

Exascale systems are predicted to have approximately one billion cores, assuming Gigahertz cores. Limitations on affordable network topologies for distributed memory systems of such massive scale bring new challenges to the current parallel programming model. Currently, there are many efforts to evaluate the hardware and software bottlenecks of exascale designs. There is therefore an urgent need to model application performance and to understand what changes need to be made to ensure extrapolated scalability. The fast multipole method (FMM) was originally developed for accelerating N -body problems in astrophysics and molecular dynamics, but has recently been extended to a wider range of problems, including preconditioners for sparse linear solvers. Its high arithmetic intensity combined with its linear complexity and asynchronous communication patterns makes it a promising algorithm for exascale systems. In this paper, we discuss the challenges for FMM on current parallel computers and future exascale architectures, with a focus on inter-node communication. We develop a performance model that considers the communication patterns of the FMM, and observe a good match between our model and the actual communication time, when latency, bandwidth, network topology, and multi-core penalties are all taken into account. To our knowledge, this is the first formal characterization of inter-node communication in FMM, which validates the model against actual measurements of communication time.

1 Introduction

N -body problems arise in many areas of physics (e.g. astrophysics, molecular dynamics, acoustics, electrostatics). In these problems, the system is described by a set of N particles and the dynamics of the system arise from interactions that occur between every pair of particles. This requires $\mathcal{O}(N^2)$ computational complexity. For this reason, many efforts have been directed at producing *fast* N -body algorithms. More efficient algorithms of the particle interaction problem can be provided by a hierarchical approach using tree structures. In this approach, the computational domain is hierarchy subdivided, and the particles are clustered into a hierarchical tree structure. The approximation is applied to far-field interactions, whereas near-field interactions are summed directly. When the far-field expansion is calculated against the particles directly, this approach called a treecode [1]. When the far-field effect is translated to local-expansions before summing their effect, it is called a fast multipole method (FMM) [13, 4]. These approaches bring the complexity down to $\mathcal{O}(N \log N)$

and $\mathcal{O}(N)$ for treecode and FMM, respectively. FMM has been listed as one of the top ten algorithms of the twentieth century [7] due to its wide applicability and impact on scientific computing. It was originally developed for applications in electrostatics and astrophysics, but continues to find new areas of application such as aeroacoustics [27], fluid dynamics [12], magnetostatics [26], and electrodynamics [29]. Because of its linear complexity, FMMs scale well with respect to the problem size, if implemented efficiently.

Since the performance of a single-processor core has plateaued, future supercomputing performance will depend mainly on increases in system scale rather than improvements in single-processor performance. Processor counts are now going from hundreds of thousands to millions, which means that the number of cores, interconnect, and memory will grow enormously. For this reason, modeling application performance at these scales and understanding what changes need to be made to ensure continued scalability on future exascale architectures is necessary. Since the performance of the FMM has a large impact on a wide variety of applications across a wide range of

disciplines, it is important to understand the challenges that FMMs face on future architectures with increased parallelism, as well as to predict and locate bottlenecks that might cause performance degradation.

The present study develops and demonstrates a performance model for the communication in FMM. To model the performance, we start with the baseline model which is the basic $\alpha - \beta$ model for communication, where α is the latency and β is the inverse bandwidth. Then, some penalties are added to the baseline model based on machine constraints. These penalties include distance and reduced per-core bandwidth. We validate our performance model on three different architectures, Shaheen (BG/P), Mira (BG/Q), and Titan (Cray XK7).

The paper is organized as follows. Section 2 gives an overview of related work. Section 3 summarizes some performance challenges that face FMM on parallel machines. These challenges include massive parallelism and degradation due to inter-node communication. In Section 4, an exposition of the fast multipole method sufficiently detailed to expose communication properties is given. Section 5 describes our performance model. Experiments done to validate the performance models are provided in Section 6. Lessons from the model results are presented in Section 7 and we conclude in Section 8.

2 Related work

Performance modeling and characterization for understanding and predicting the performance of scientific applications on HPC platforms has been targeted by many related projects. For example, Clement and Quinn developed a performance prediction methodology through symbolic analysis of their source code [5]. Mendes and Reed focused on predicting scalability of an application program executing on a given parallel system [22]. Mendes proposed methodology to predict the performance scalability of data parallel applications on multi-computers based on information collected at compile time [21]. The approach of combining computation and communication to obtain a general performance model is described by Snaveley *et al.* [25]. DeRose and Reed concentrate on tool development for performance analysis [6]. Performance models for a specific given application domain, which presents performance bounds for implicit CFD codes have also been considered [15]. The efficiency of the spectral transform method on parallel computers has been evaluated by Foster [9]. Kerbyson *et al.* provide an analytical model for the application SAGE

[17]. Performance models for AMG were developed by Gahvari *et al.* [10]. Traditional evaluation of specific machines via benchmarking is presented by Worley [28].

Scaling FMM to higher and higher processors counts has been a popular topic [23, 16], while extensive study of single-node performance optimization, tuning, and analysis of FMM has also been of interest [3]. However, there has been little effort to model the inter-node communication of FMMs. Lashuk *et al.* derive the overall complexity of FMM on distributed memory heterogeneous architectures [18], but do not validate the model against the actual performance. The present work is based on the communication model for AMG [10], and extends their theory to FMM. To our knowledge, this is the first formal characterization of inter-node communication in FMM, which validates the model against actual measurements of communication time.

3 Performance challenges

High performance computing systems have shown a fast and sustained growth with performance improvement of 10x every 3.6 years. This performance improvement comes at a high cost and introduces many challenges. Furthermore, the development of an exascale computing capability will cause significant and dramatic changes in computing hardware architecture relative to current petascale computers. In this section we present some of the challenges faced by FMMs to achieve good parallel performance on future exascale systems.

3.1 Trends in Computer Hardware

Massively parallel machines have emerged as the most widely used high-performance computing platforms. These machines are characterized currently by hundreds of thousands of computing nodes, and this number will continue to grow. Another trend of massively parallel machines is to increase the numbers of cores on each node. These nodes communicate by sending messages through a network, which leads to lower scalability and less performance due to cores on a single node contending for access to the interconnect. We discuss multicore issues in more detail when presenting our performance models that take this into account.

3.2 Communication

Algorithms have two costs in terms of time and energy: computation (flops) and communication

Table 1: Differential rates of improvement of computation and communication

	Annual Improvements		
<i>floprate</i>		<i>bandwidth</i>	<i>latency</i>
59%	Network	26%	15%
	DRAM	23%	5%

(Bytes). Communication involves moving data between levels of a memory hierarchy in case of sequential algorithms and exchanging data between processors over a network in the case of parallel algorithms. Therefore, without considering overlap, the running time of an algorithm is the sum of three terms: the number of flops times the time per flop, the number of words moved divided by the bandwidth (measured as words per unit time), and the number of messages times the latency. The last two terms are time consumed by communication. Given that the time per flop is generally much less than the reciprocal of bandwidth for most applications, we can see that inter-node communication eventually consumes most of the running time for large scale calculations. Furthermore, the gaps between computation and communication are growing exponentially with time, as we can see in Table 1.

Communication latency and bandwidth are becoming bottlenecks, where the latency is governed by physics and the bandwidth by cost. Therefore, minimizing communication within the algorithm becomes crucial. Appropriate performance models can guide development of algorithms to help reduce the communication.

4 Fast multipole method

N -body methods are most commonly used to simulate the interaction of particles in a potential field, which has the form

$$f(\mathbf{x}_i) = \sum_{j=1}^N q_j K(\mathbf{x}_i, \mathbf{x}_j) \quad (1)$$

Here, $f(\mathbf{x}_i)$ represents a field value evaluated at a point \mathbf{x}_i which is generated by the influence of sources located at \mathbf{x}_j with weights q_j . $K(\mathbf{x}_i, \mathbf{x}_j)$ is the kernel that governs the interactions between evaluation and source particles. The direct approach to simulate the N -body problem is relatively simple; it evaluates all pair-wise interactions among the particles. While this method is exact to within machine precision, the solution is $\mathcal{O}(N^2)$ in its computational complexity, which

is prohibitively expensive for even modestly large data sets. However, its simplicity and ease of implementation make it an appropriate choice when simulating small particle sets ($N < 100$) where high accuracy is desired [24]. For a larger number of particles, many faster algorithms have been invented, e.g. treecode and fast multipole method (FMM). The main idea behind these fast algorithms is to approximate the effect of sufficiently far particles. The most common way to achieve this approximation is to cluster the far particles into larger and larger groups by constructing a tree structure. The treecode clusters the far particles and achieves $\mathcal{O}(N \log N)$ complexity. The FMM further clusters the near particles in addition to the far particles to achieve $\mathcal{O}(N)$ complexity.

In this section, we present an overview of fast algorithms that have been developed for the calculation of N -body problems. First, the spatial hierarchy and the fast approximate evaluation of these algorithms are discussed. Then, a description of the communication introduced by the domain partitioning scheme used in these algorithms is provided. The main focus is on the data flow of the FMM algorithm for which we develop the performance model.

4.1 FMM Overview

This overview is intended to introduce some key ingredients of the FMM. The mathematics behind the specific FMM kernels is outside the scope of this study, since it has very little to do with the communication model. For details of the mathematics we refer the reader to previous publications on FMM [2, 4].

4.1.1 Basic Component

Both treecode [1] and FMM [13] are based on two key ideas: the tree representation for the spatial hierarchy, and the fast approximate evaluation. The spatial hierarchy means that the computational domain is hierarchically decomposed into increasing levels of refinement, and then the near and far subdomains can be identified at each level. The three-dimensional spatial domain of the treecode and FMM is represented by octrees, where the space is recursively subdivided into eight cells until the finest level of refinement or “leaf level”. Figure 1 illustrates such a hierarchical space decomposition for a two-dimensional domain (a), associated to a quad-tree structure (b). The original FMM [14] is based on a series expansion of the Laplace Green’s function ($1/r$) and therefore can be applied to the evaluation of related potentials and/or forces [11]. The approximation reduces the number of operations in exchange for accuracy.

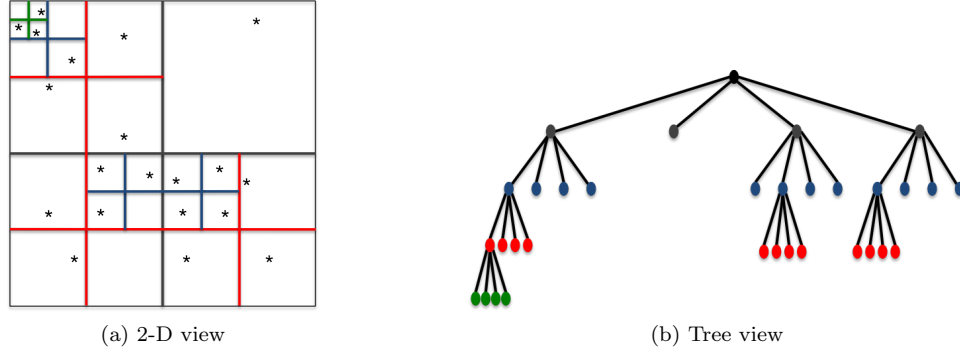


Figure 1: Hierarchical decomposition

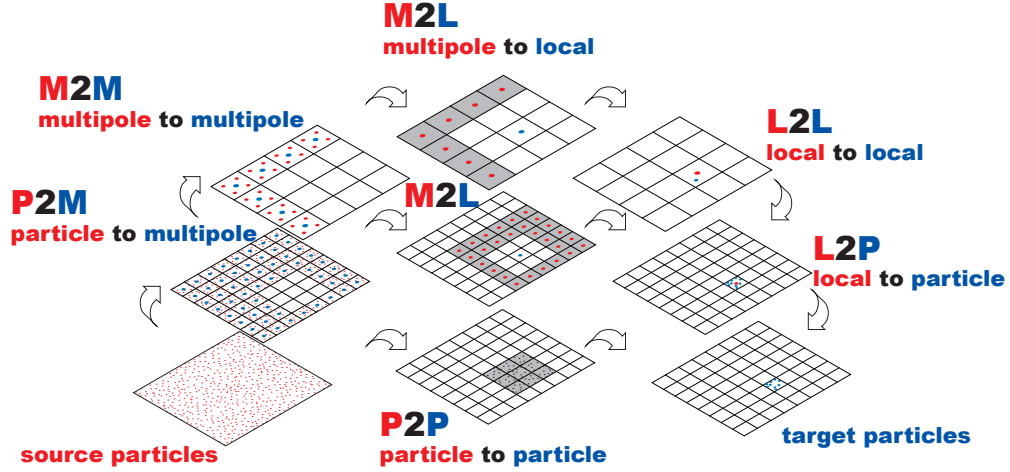


Figure 2: Data-flow of FMM calculation. Data dependency is between red and blue points.

4.1.2 Flow of Calculation

Figure 2, shows the flow of FMM where the effect of the source particles, shown in red in the lower left corner, are calculated on the target particles, shown in blue in the lower right corner. The schematic is a 2-D representation of what is actually a 3-D octree structure. The calculation starts by transforming the mass/charge of the source particles to a multipole expansion (P2M). Then, the multipole expansion is translated to the center of larger cells (M2M). Then, the influence of multipoles on the particles is calculated in three steps. First, it translates the multipole expansion to a local expansion (M2L). Next, the center of expansion is translated to smaller cells (L2L). Finally, the effect of the local expansion in the far field is translated onto the target particles (L2P). All pairs interaction is used to calculate the effect of near field on target particles (P2P).

4.2 FMM Communication Scheme

Partitioning of the FMM global tree structure and communication stencils is shown in Figure 3. The binary tree on the left side is a simplification of what is actually an octree in a 3-D FMM. Likewise, the schematics on the right are a 2-D representation of what is actually a 3-D grid structure. Each leaf of the global tree is a root of a local tree in a particular MPI process, where the global tree has L_{global} levels, and the local tree has L_{local} levels. Each process stores only the local tree, and communicates the halo region at each level of the local and global tree as shown in the red hatched region in the four illustrations on the right. The blue, green, and black lines indicate global cell boundaries, process boundaries, local cell boundaries, respectively. The switch between local and global trees produces a change in the communication pattern as shown in Figure 4.

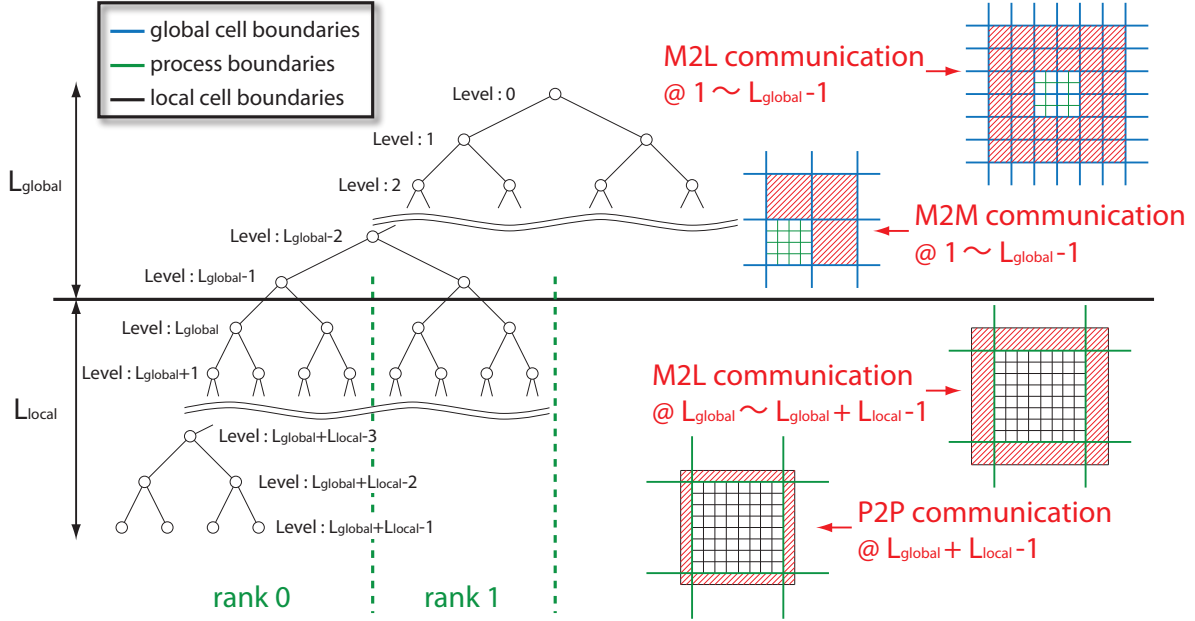


Figure 3: Splitting of the local and global tree in FMM.

5 Modeling Performance

Performance modeling is a key ingredient in high performance computing. It has a great importance in the design, development and optimization of applications, architectures and communication systems. It also plays a crucial role in understanding important performance bottlenecks of complex systems. For this reason, performance models are used to analyze, predict, and calibrate performance for systems of interest. In this section we develop a performance model to understand the performance of FMM and to make predictions on future machines.

First, we start with the baseline model that is a combination of the latency and inverse bandwidth. We subsequently refine this baseline model to reach a complete model that is able to cover the relevant system architecture properties, with the exception that overlapping communication with computation is not considered in this work.

5.1 FMM Communication Phases

As shown in Figure 3, our FMM uses a separate tree structure for the local and global tree. In order to construct a performance model for the communication in FMM, we estimate the amount of data that must be sent at each level of the hierarchy. Table 2 shows the number of cells that are sent, which correspond to the illustrations in Figure 3. L_{global} is the depth of the global tree, L_{local} is the depth of the local tree. We define N as the global number of particles, and P as the number of processes (MPI ranks). The global tree is constructed so that each MPI process is a leaf node in the global tree. Therefore, the depth of the global tree only depends on the number of processes P and not N . The depth of the global tree grows with $\log_8 P$, whereas the depth of the local tree grows with $\log_8(N/P)$. For the current calculations we are assuming a nearly uniform particle distribution (as in explicit solvent molecular dynamics) and therefore a full octree structure.

5.1.1 Global M2L

In Table 2 we show the amount of cells to send per level and the total amount of communication for all levels. There are four types of communication in our FMM, which correspond to the four stages shown with the red hatching in Figure 3. The first is the “Global M2L” communication, which sends 26×8 cells at each level, as shown at the top right of Figure

Table 2: Amount of communication in FMM

	Cells to send / level	Total comm.
Global M2L	26×8	$\mathcal{O}(\log P)$
Global M2M	7	$\mathcal{O}(\log P)$
Local M2L	$(2^i + 4)^3 - 8^i$	$\mathcal{O}((N/P)^{2/3})$
Local P2P	$(2^i + 2)^3 - 8^i$	$\mathcal{O}((N/P)^{2/3})$

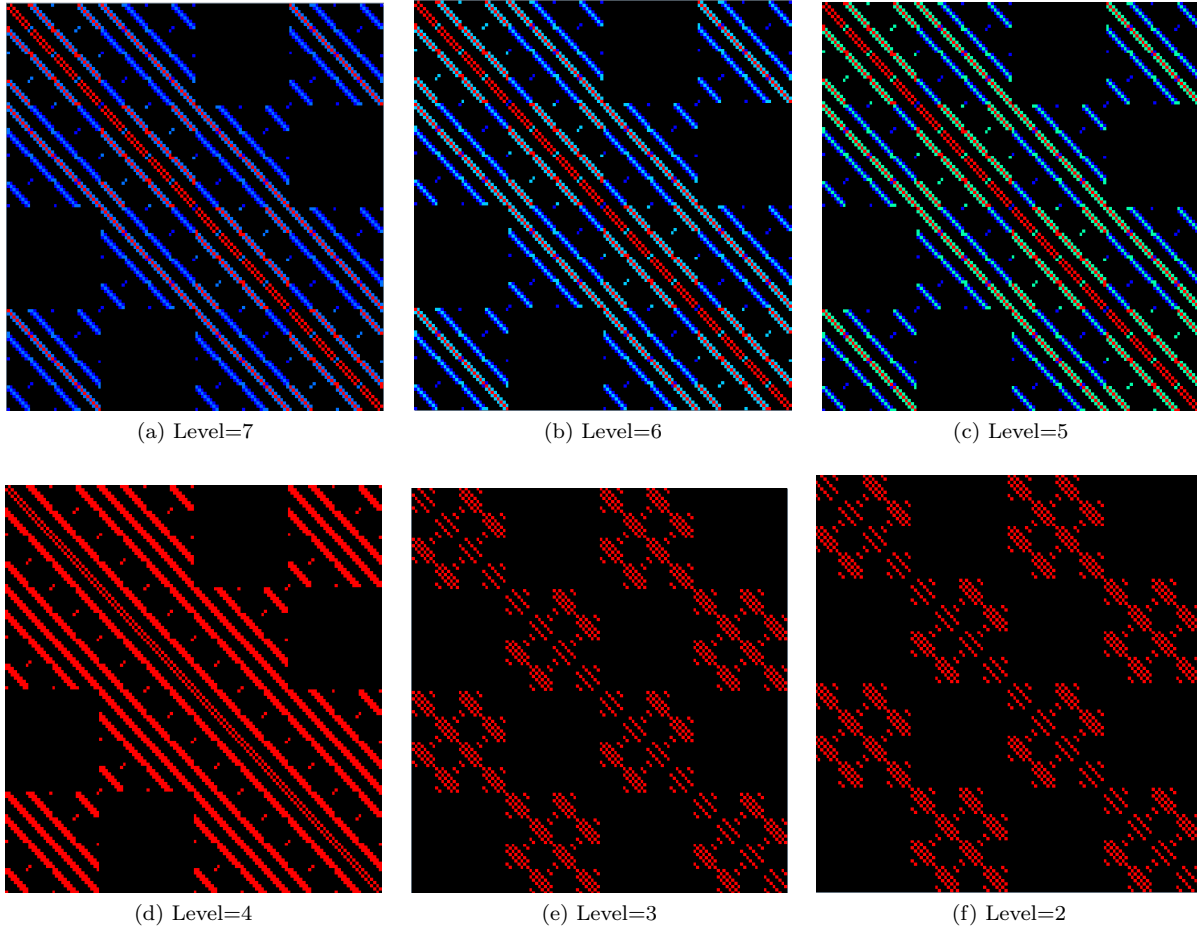


Figure 4: Level-by-level communication patterns for the M2L phase of an FMM with $N=62,500$ per process using 128 processes. Areas of black indicate zero messages between processes.

3. The green lines are the process boundaries and the blue lines are the cell boundaries, which means one FMM cell belongs to many processes in the global tree. In order to avoid redundant communication, we index each process that shares a global cell and perform a one-to-one communication between the processes with matching indices only. In order to further reduce the communication, we select one process for a group of eight cells to do the communication. Therefore, the number of processes to communicate with (p_i) is always 26 and the number of cells to send is always 8 for every process and for every level in the global tree. In other words, for the “Global M2L” communication the message size and number of sends is constant regardless of N and P , and only the number of hops between the processes will increase depending on the network topology. On torus networks, we map the MPI ranks to the torus and synchronize the direction of the 26 one-to-one communications. The communication per level is $\mathcal{O}(1)$ and the number

of levels in the global tree is $\mathcal{O}(\log P)$, so the total communication complexity for this stage is $\mathcal{O}(\log P)$ as shown in Table 2.

5.1.2 Global M2M

The second type of communication is the “Global M2M”, which sends 7 cells at each level, as shown in Figure 3. We use a similar technique to the “Global M2L” case to avoid redundant communication by pairing the MPI ranks for the one-to-one communication when many processes share the same global cell. The number of processes to communicate with is always seven and the number of cells to send is always one for every process and for every level in the global tree. Similar to the “Global M2L” case, only the number of hops during the one-to-one communication will increase, and the rate depends on the network topology. The communication per level is $\mathcal{O}(1)$ and the number of levels is $\mathcal{O}(\log P)$, so the total communi-

cation is $\mathcal{O}(\log P)$ for the “Global M2M” stage.

5.1.3 Local M2L

The third type of communication is the “Local M2L”, which is shown in the red hatching in the second picture from the bottom on the right side of Figure 3. The process boundaries shown in green are coarser than the local cell boundaries shown in black, which means that one process contains many cells contrary to the previous two communication types. In a full octree structure, we know that all cells are non-empty so we simply need to send two layers of halo cells for the M2L calculation at each level, as shown in Figure 3. Therefore, the number of processes to communicate with is always the 26 neighbors, and the number of cells to send depends on the level. At level i of the local tree, there are 2^i cells in each direction. Two layers of halo cells on each size will create a volume of $(2^i + 4)^3$ cells, and subtracting the center volume 8^i will give $(2^i + 4)^3 - 8^i$ as shown in Table 2. The leading term is $\mathcal{O}(4^i)$ since the 8^i term cancels out. Since the number of levels in the local tree grow as $\log_8(N/P)$ the communication complexity for the “Local M2L” is $\mathcal{O}(4^{\log_8(N/P)}) = \mathcal{O}((N/P)^{2/3})$. This can also be understood as the surface to volume ratio of the bottom two illustrations in Figure 3. Since N/P is constant for weak scaling and decreases for strong scaling, this part does not affect the asymptotic weak/strong scalability of the FMM.

5.1.4 Local P2P

The fourth type of communication in the FMM is the “Local P2P”, which is shown in the bottom picture on the right side of Figure 3. This communication only happens at the bottom level of the local tree. Similar analysis to the “Local M2L” stage shows that $(2^i + 2)^3 - 8^i$ cells must be sent, as shown in Table 2. In this case, i is exactly $\log_8(N/P)$ and we obtain the same asymptotic amount of communication of $\mathcal{O}((N/P)^{2/3})$. Similar to the “Local M2L”, this part does not affect the asymptotic weak/strong scalability of the FMM. However, the content of the data is different from the previous three cases where the multipole expansion coefficients were being sent. In the P2P communication the coordinates and the charges of every particle that belongs to the cell must be sent. Therefore, the asymptotic constant of $\mathcal{O}(N/P)^{2/3}$ is typically much larger than that of the “Local M2L”, and this could be the dominant part of the communication time depending on the number of particles per leaf cell.

5.2 Baseline Model ($\alpha - \beta$ model)

To model interprocess communication, we start by the basic $\alpha - \beta$ model, where α represents communication latency, where β is the send time per-Byte (inverse bandwidth). Using the basic $\alpha - \beta$ model, a message send cost can be represented as

$$T_{\alpha-\beta} = \alpha + n\beta \quad (2)$$

where n is the number of Bytes in the message.

This basic model describes the communication over an ideal architecture where the communication cost does not depend on processor locations or network traffic caused by many processors communicating at the same time [8]. For a more realistic architecture, a more detailed model is needed. For this reason, we add penalties to this basic model to take into account machine-specific performance issues. In particular, we consider communication distance, interconnection switching delay, limited bandwidth, and the effect of multiple cores on a single node contending for available resources.

5.3 Distance Penalty ($\alpha - \beta - \gamma$ Model)

Following [10], we refine the assumption that distance between processors in interconnected networks does not have effect on communication time. To take into account the effect of distance we refine the baseline model according to the number of extra hops a message travels

$$T_{\alpha-\beta-\gamma} = \alpha + n\beta + (h - h_m)\gamma, \quad (3)$$

where h is the number of hops a message travels, h_m is the smallest possible number of hops a message can travel in the network, and γ is the delay per extra hop. If there is no network contention and all messages travel with minimum number of hops, this distance penalty should have no effect.

5.4 Bandwidth Penalty on β

The peak hardware bandwidth is rarely achieved in message passing. Therefore, we multiply β by B_{max}/B to incorporate the ratio between the peak hardware per-node bandwidth B_{max} and the effective bandwidth from the benchmark B .

$$T_{\beta-Penalty} = \alpha + n\beta \frac{B_{max}}{B} + (h - h_m)\gamma \quad (4)$$

5.5 Multicore Penalty on α or γ

Increasing the number of cores per node increases the data traffic between nodes, and could potentially result in congestion. Furthermore, larger number of

cores per node introduces more noise caused by access to resources shared by multiple cores. To model these effects, we multiply α and/or γ by the number of active cores per node c . This model focuses on the worst case behavior where a machine’s aggregate bandwidth could be exceeded by all cores communicating simultaneously. The resulting models are

$$T_{\alpha-Penalty} = c\alpha + n\beta + (h - h_m)\gamma \quad (5)$$

$$T_{\gamma-Penalty} = \alpha + n\beta + c(h - h_m)\gamma \quad (6)$$

6 Model Validation

6.1 Machine Description

To validate our performance models, we benchmark our FMM code on three different architectures; Shaheen, Mira, and Titan.

Shaheen is 16 racks of an IBM BlueGene/P. Each rack contains 1024 PowerPC 450 CPUs with 4 cores running at 850MHz with 32kB private L1 cache and 8MB shared L3 cache. Each compute node has 2GB RAM with 13.6 GB/s memory bandwidth. The nodes are connected by 3-D torus network with 5.1GB/s injection bandwidth per node.

Mira is 48 racks of an IBM BlueGene/Q. Each rack contains 1024 Power A2 CPUs with 16 + 1 cores running at 1.6GHz with 16kB private L1 cache and 32MB shared L2 cache. Each compute node has 16GB RAM with 42.6GB/s memory bandwidth. The nodes are connected by a 5-D torus network with 20GB/s injection bandwidth per node.

Titan is a Cray XK7 system with 18,688 compute nodes each equipped with an AMD Opteron 6274 CPU and NVIDIA Kepler K20X GPU. The CPU has 16 cores running at 2.2 GHz with 16 kB L1 cache, 2 × 4 MB L2 cache, and 8 × 2 MB L3 cache. The GPU has 15 × 64 cores running at 730 MHz with 64 + 48 kB L1 cache and 1.5 MB L2 cache. Each compute node has 32 GB of RAM with 51.2 memory bandwidth. The nodes are connected by a 3-D torus with 20GB/s of injection bandwidth per node. We do not use any of the GPUs in the current study.

In order to obtain the machine parameters, the **b_eff** benchmark in the HPC Challenge suite [20] was used to determine the parameters α and β . We report the best-case latency and bandwidth measurements. To find the parameter γ , we followed the same procedure as Gahvari *et al.* [10]. The machine parameters for Shaheen, Mira, and Titan are shown in Table 3. Note that our definition of β is defined as send time per Byte, whereas Gahvari *et al.* define their β as send time per element (8 Bytes).

Table 3: Machine parameters for latency α , inverse bandwidth β , and distance penalty γ , on Shaheen, Mira, and Titan.

	Shaheen	Mira	Titan
α	4.12 μs	5.33 μs	1.67 μs
β	2.14 ns	1.32 ns	1.62 ns
γ	29.9 ns	134 ns	284 ns

6.2 Experimental Setup

We ran the FMM code for 10 steps and measured the time spent on the communication for the “Global M2L” and “Local M2L” phases. The results are then divided by 10 to get the average time spent at each level. The “Global M2M” phase was negligible and the “Local P2P” phase only occurs at the bottom level and is irrelevant to the scalability of the FMM, so we do not consider these two phases in the current analysis. We used the Laplace kernel in three dimensions with random distribution of particles in a cube. We use periodic boundary conditions so that there is no load imbalance at the edges of the domain. The number of MPI processes was varied between $P = \{128, 1024, 8192\}$, while the number of particles per process was kept constant at $N/P = 62, 500$.

Table 4 shows communication information and statistics when running the FMM on 128, 1024, and 8192 processes. “Level” is the level within the tree structure and goes from 0 to $L_{global} + L_{local} - 1$, where $L_{local} = 4$ for $N/P = 62, 500$. Therefore, the bottom four levels in Table 4 (a), (b), and (c) belong to the local tree. The depth of the global tree L_{global} is 4, 5, and 6 for 128, 1024, and 8192 processes, respectively. “Cells” is the total number of cells at that level of the tree structure, which is simply 8^{Level} for a full octree. “Sends” is the number of processes that each processes sends to. As mentioned in Section 5.1 we have developed a communication scheme that limits the number of sends to 26 regardless of the problem size, number of processes, or the level. “Bytes” is the aggregate data size that is sent by a given process at each level of the tree. As shown in Table 2, the number of cells for the “Global M2L” communication is 26×8 . For each cell we are sending 56 multipole expansion coefficients in single precision (4 Bytes). Therefore, the total number of Bytes for the “Global M2L” phase is $26 \times 8 \times 56 \times 4 = 46592$. We can see from Table 2 that the amount of cells involved in the “Local M2L” communication can be calculated by $(2^i + 4)^3 - 8^i$, where i is the level in the local tree (not the “Level” shown in Table 4). For example, for level one in the local tree, the amount of cells will be $(2^1 + 4)^3 - 8^1$ which is equivalent to 26×8 . This is

Table 4: Statistics of the M2L communication.

(a) 128 Processes

Level	Cells	Sends	Bytes
0	1	0	0
1	8	0	0
2	64	26	46592
3	512	26	46592
4	4096	26	46592
5	32768	26	100352
6	262144	26	272384
7	2097152	26	874496

(b) 1024 Processes

Level	Cells	Sends	Bytes
0	1	0	0
1	8	0	0
2	64	26	46592
3	512	26	46592
4	4096	26	46592
5	32768	26	46592
6	262144	26	100352
7	2097152	26	272384
8	16777216	26	874496

(c) 8192 Processes

Level	Cells	Sends	Bytes
0	1	0	0
1	8	0	0
2	64	26	46592
3	512	26	46592
4	4096	26	46592
5	32768	26	46592
6	262144	26	46592
7	2097152	26	100352
8	16777216	26	272384
9	134217728	26	874496

why the “Bytes” is the same for the “Global M2L” and the first level of the “Local M2L” in Table 4.

6.3 Model Validation

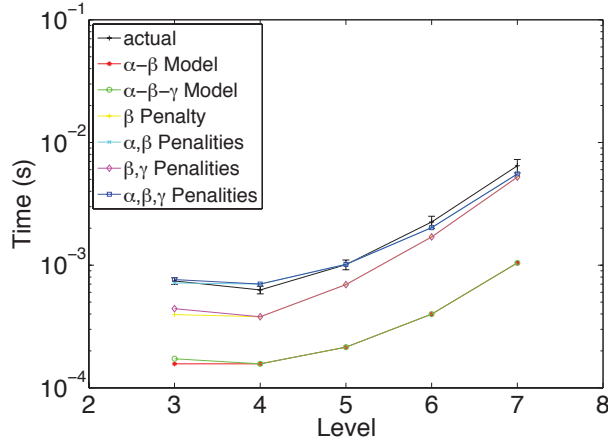
We compare the actual communication time for the M2L communication with our performance model on Shaheen, Mira, and Titan. We compare against same combination of models as in the multigrid study [10]. The combinations are:

1. Baseline model ($\alpha - \beta$ model)
2. With distance penalty ($\alpha - \beta - \gamma$ model)
3. With distance and bandwidth penalty (β penalty)

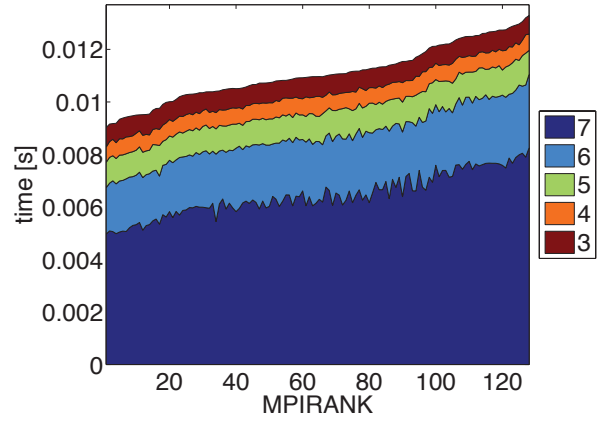
4. With distance and bandwidth penalty, plus multicore penalty on latency (α, β penalty)
5. With distance and bandwidth penalty, plus multicore penalty on distance (β, γ penalty)
6. With distance and bandwidth penalty, plus multicore penalty on latency and distance (α, β, γ penalty)

The results on Shaheen are shown in Figure 5. The actual measured performance is shown as a black line, where an error bar is drawn according to the standard deviation in communication time among the different MPI ranks. By comparing the Bytes in Table 4 with the communication time in Figure 5, we see that the deepest four levels that belong to the “Local M2L” phase have a communication time that is proportional to the data size being sent. The main discrepancy in the models is caused by the β penalty, for which the ratio between the theoretical injection bandwidth and the `b_eff` benchmark results is accounted for. The actual communication time agrees well with the models with α , β , and γ penalties. For the shallow levels that belong to the “Global M2L” phase, the communication time increases as the level decreases/coarsens. The reason for this can be understood by looking back at Figure 3, where the “Global M2L” is communicating with farther processes at coarser levels of the tree. Since we are mapping the geometric partitioning of the octree to the 3-D torus network of Shaheen, the proximity in the octree directly translates to the proximity in the network. Therefore, even though the data size is constant for all levels in the “Global M2L” phase, the number of hops is larger, which accounts for switching delays and also network contention to some extent. This increases the communication time at coarser levels and the models that incorporate γ are able to predict this behavior.

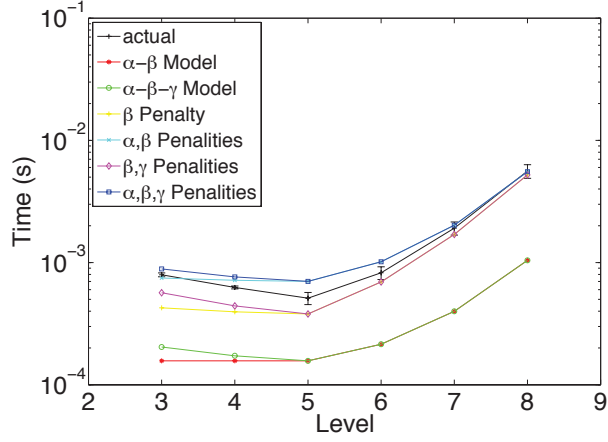
In Figure 6, the M2L communication time on Shaheen is plotted against the MPI rank to show the load balance between the processes. Each color shows M2L communication at a different level of the tree structure, and the numbers in the legend represent the levels. The communication time of each level is stacked on top of each other so that the total height of the area plot represents the total M2L communication time shown in Figure 5. The MPI ranks are sorted according to the total M2L communication time for better visibility in the small differences between processes. As can be seen from the figure, the load balance is quite good. The imbalance seems to come from the finest levels, which are 7, 8, and 9 for 128, 1024, and 8192 processes, respectively.



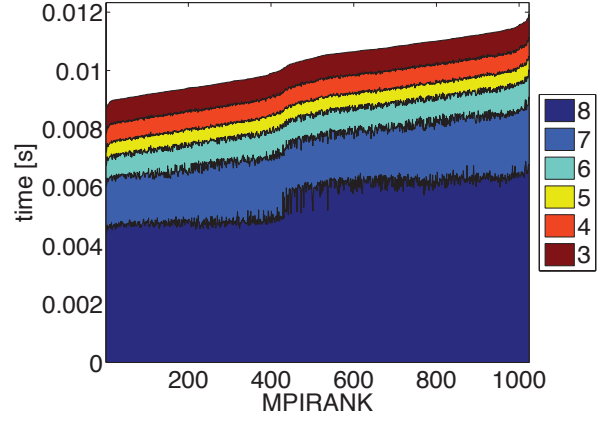
(a) 128 processes



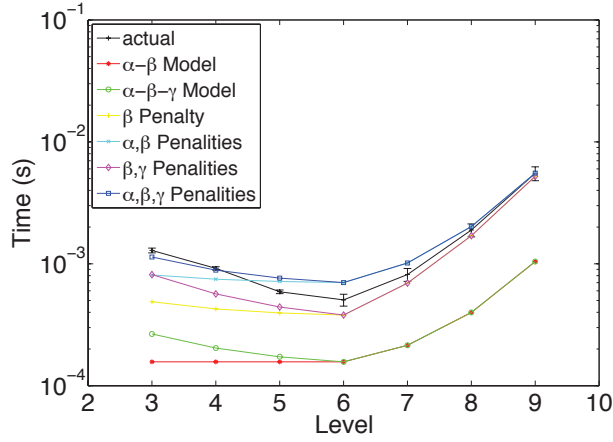
(a) 128 processes



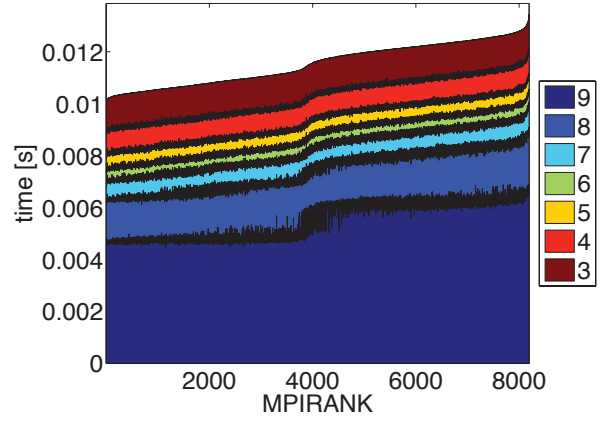
(b) 1024 processes



(b) 1024 processes



(c) 8192 processes



(c) 8192 processes

Figure 5: Performance model prediction and actual time for M2L communication phase on Shaheen.

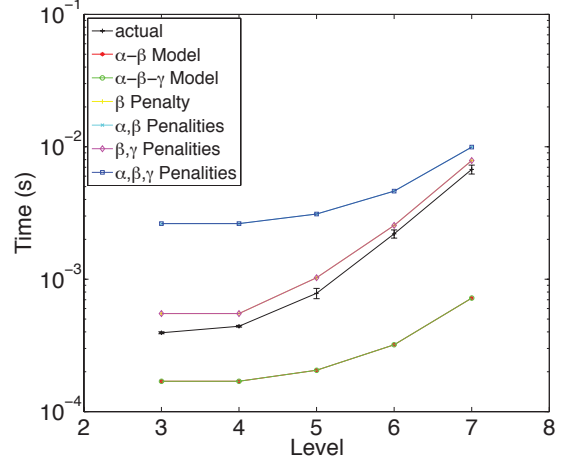
Figure 6: Load balance of M2L communication phase on Shaheen.

The M2L communication time on Mira is plotted along with the six model predictions in Figure 7. Similar to the runs on Shaheen, the main difference in the model predictions is caused by the β penalty. We also see a discrepancy between the model predictions with and without the α penalty for the “Global M2L” phase (coarser levels). The multicore penalty is very small on the Bluegene/Q, so the model without the multicore penalty show a better agreement in terms of slope of the curve. For example “ β Penalty” and “ β Penalty” match quite well with the actual time. This lack of multicore penalty has been observed in other applications where the use of hybrid OpenMP+MPI approach did not improve the performance over a flat MPI approach [19]. Contrary to the runs on Shaheen, the communication time has a nearly flat profile for the “Global M2L” phase. This is because the 5-D torus network minimizes the number of hops and network contention so the degradation at coarse levels of the tree is minimal. Far nodes in the octree are not so far in the Bluegene/Q network topology.

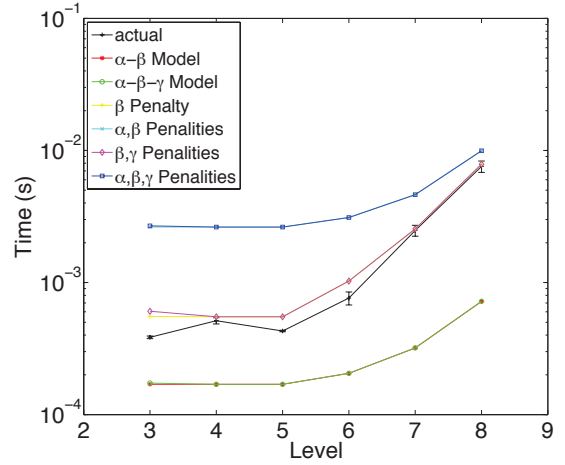
Figure 8 shows the M2L communication time on Titan along with the six model predictions. Similar to the previous two cases, the difference between the model predictions is mainly due to the correction for the inverse bandwidth. This difference in the theoretical injection bandwidth and measured effective bandwidth seems to have the largest effect on all three architectures. What is different from the previous two cases is the large jump in the actual communication time for the “Global M2L” phase. For example, for the 8192 process run level 5 is taking about 10 times more than level 6 even though the message size is 46,592 Bytes for both cases. The γ term in the current performance models anticipates such behavior. The error bars in the actual timings are quite large, which indicates that there is a large load imbalance compared to the previous two systems.

7 Conclusion

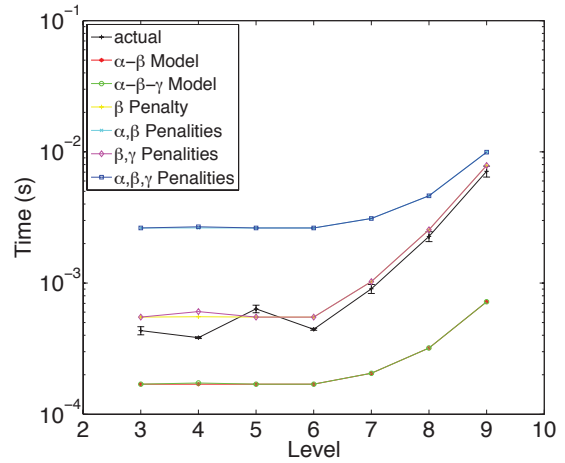
The goal of this work is to model the global communication of the FMM and anticipate challenges on future exascale machines. To improve model fidelity, we consider penalties based on machine constraints including distance effects, reduced per core bandwidth, and the number of cores per node. We observe a good match between the $\alpha - \beta - \gamma$ model with multicore penalties and the actual communication time. The discrepancy between the other models means that all components of the model; latency α , bandwidth β , hops γ , and multicore penalty must be taken into account when predicting the communica-



(a) 128 processes

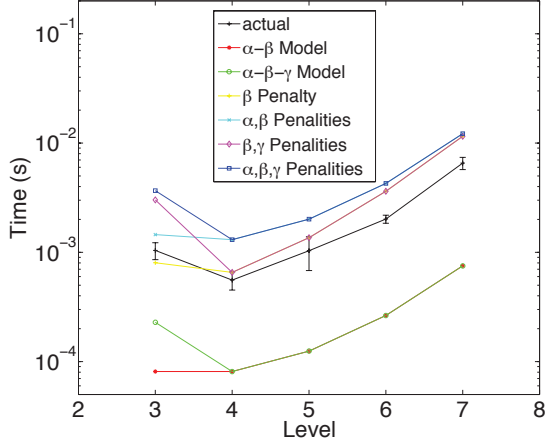


(b) 1024 processes

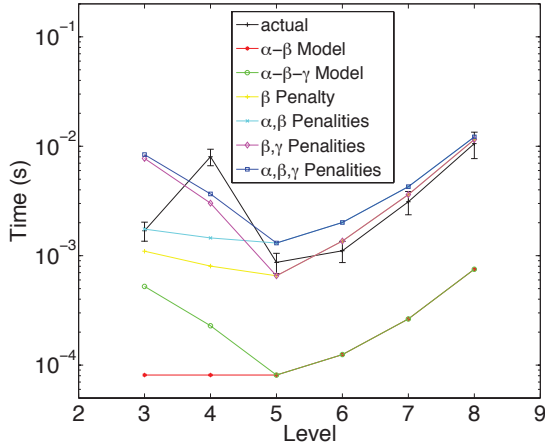


(c) 8192 processes

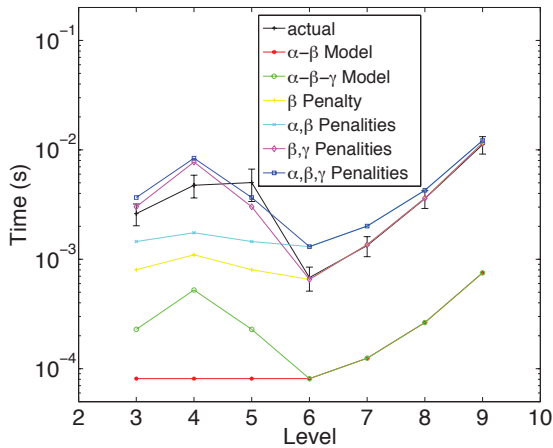
Figure 7: Performance model prediction and actual time for M2L communication phase on Mira.



(a) 128 processes



(b) 1024 processes



(c) 8192 processes

Figure 8: Performance model prediction and actual time for M2L communication phase on Titan.

tion performance of FMM.

In our benchmark tests, we compare the performance models with the actual measurements for the M2L communication, since this is the dominant part of the FMM communication. Our observations agree with that of the studies by Gahvari et al. [10] where the performance of an algebraic multigrid method is analyzed using the same model. Our measurements fall within the bounds of the performance models, and match best with the model where latency, bandwidth, hops, and multicore penalty are all taken into account.

We were able to show that the present communication model is able to predict the performance on three HPC systems with different characteristics. To our knowledge, this is the first formal characterization of inter-node communication in FMM, which validates the model against actual measurements of communication time. We believe this is a step in the right direction, and the next logical step would be to increase the number of processes and continue to benchmark new HPC systems as they become available.

Acknowledgements

The runs on Shaheen were provided through KAUST supercomputing lab. This research used resources of the Argonne Leadership Computing Facility at Argonne National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under contract DE-AC02-06CH11357. This research used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.

Author Biographies

Huda Ibeid received her BSc degree in Computer Engineering from the University of Jordan and currently getting her PhD degree in Computer Science from King Abdullah University of science and Technology (KAUST). Her research interests include fast algorithms for particle-based simulations, fast algorithms on parallel computers and GPUs, design of parallel numerical algorithms, parallel programming models and performance optimizations for heterogeneous GPU-based systems.

Rio Yokota obtained his PhD from Keio University, Japan, in 2009 and went on to work as a postdoctoral researcher with Prof. Lorena Barba at the University

of Bristol and then Boston University. He has worked on the implementation of fast N -body algorithms on special-purpose machines such as MDGRAPE-3, and then on GPUs after CUDA was released, and on vortex methods for fluids simulation. He joined the King Abdullah University of Science and Technology (KAUST) as a research scientist, where he continues to work on fast multipole methods.

David Keyes is the director of the Strategic Initiative for Extreme Computing at KAUST and an affiliate of several laboratories of the U.S. Department of Energy (DOE). Keyes graduated in Aerospace and Mechanical Sciences from Princeton University and earned a doctorate in Applied Mathematics from Harvard University. He did postdoctoral work in the Computer Science Department of Yale University. With backgrounds in engineering, applied mathematics, and computer science, he works at the algorithmic interface between parallel computing and the numerical analysis of partial differential equations, across a spectrum of aerodynamic, geophysical, and chemically reacting flows.

References

- [1] J. Barnes and P. Hut. $O(N \log N)$ force-calculation algorithm. *Nature*, 324:446–449, 1986.
- [2] R. Beatson and L. Greengard. A short course on fast multipole methods. In *Wavelets, Multilevel Methods and Elliptic PDEs*, pages 1–37. Oxford Science Publications, 1997.
- [3] A. Chandramowlishwaran, S. Williams, L. Oliker, I. Lashuk, G. Biros, and R. Vuduc. Optimizing and tuning the fast multipole method for state-of-the-art multicore architectures. In *Proceeding of the International Parallel Distributed Processing Symposium (IPDPS)*, pages 1–12, 2010.
- [4] H. Cheng, L. Greengard, and V. Rokhlin. A fast adaptive multipole algorithm in three dimensions. *Journal of Computational Physics*, 155(2):468–498, 1999.
- [5] M. J. Clement and M. J. Quinn. Symbolic performance prediction of scalable parallel programs. In *Proceedings of the International Parallel Processing Symposium*, pages 635–639, April 1995.
- [6] L. DeRose and D. A. Reed. Svpablo: A multi-language, architecture-independent performance analysis system. In *Proceeding of the International Conference on Parallel Processing*, pages 311–318, August 1999.
- [7] J. Dongarra and F. Sullivan. Guest Editors Introduction to The Top 10 Algorithms. *Computing in Science and Engineering*, 2:22–23, 2000.
- [8] I. Foster. *Designing and Building Parallel Programs*. Addison-Wesley, 1995.
- [9] I. T. Foster and P. H. Worley. Parallel algorithms for the spectral transform method. *SIAM Journal on Scientific and Statistical Computing*, 18(3):806–837, 1997.
- [10] H. Gahvari, A. H. Baker, M. Schulz, U. M. Yang, K. E. Jordan, and W. Gropp. Modeling the performance of an algebraic multigrid cycle on HPC platforms. In *ICS '11 Proceedings of the International Conference on Supercomputing*, pages 172–181, 2011.
- [11] N. L. Gorn and D. V. Berkov. Adaptation and performance of the fast multipole method for dipolar systems. *Journal of Magnetism and Magnetic Materials*, 272-276:698–700, 2004.
- [12] L. Greengard, M. C. Kropinski, and A. Mayo. Integral equation methods for stokes flow and isotropic elasticity in the plane. *Journal of Computational Physics*, 125:403–414, 1996.
- [13] L. Greengard and V. Rokhlin. A fast algorithm for particle simulations. *Journal of Computational Physics*, 73(2):325–348, 1987.
- [14] L. Greengard and Rokhlin V. On the efficient implementation of the fast multipole algorithm. Research Report RR-602, Yale University, 1988.
- [15] W. D. Gropp, D.K. Kaushik, D.E. Keyes, and B.F. Smith. Toward realistic performance bounds for implicit CFD codes. In *Proceedings of Parallel CFD'99*, pages 23–26, May 1999.
- [16] P. Jetley, L. Wesolowski, F. Gioachin, L. V. Kale, and T. R. Quinn. Scaling hierarchical N-body simulations on GPU clusters. In *SC '10 Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–11, 2010.
- [17] D. Kerbyson, H. Alme, A. Hoisie, F. Petrini, A. Wasserman, and M. Gittings. Predictive performance and scalability modeling of a large-scale application. In *Proceedings of the 2001 ACM/IEEE conference on Supercomputing*, pages 1–12, 2001.
- [18] I. Lashuk, A. Chandramowlishwaran, H. Langston, T.-A. Nguyen, R. Sampath, A. Shringarpure, R. Vuduc, L. Ying, D. Zorin, and G. Biros. A massively parallel adaptive fast multipole method on heterogeneous architectures. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, pages 1–12, 2009.
- [19] M. Lee, N. Malaya, and R. D. Moser. Petascale direct numerical simulation of turbulent channel flow on up to 768k cores. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, Denver, CO, USA, November 16-22 2013.
- [20] P. Luszczek and J. Dongarra. Introduction to the HPCChallenge Benchmark Suite. Technical Report

ICL-UT-05-01, university of Tennessee, Knoxville, March 2005.

- [21] C. L. Mendes. *Performance Scalability Prediction on Multicomputers*. Phd thesis, University of Illinois, Urbana-Champaign, May 1997.
- [22] C. L. Mendes and D. A. Reed. Integrated compilation and scalability analysis for parallel systems. *International Conference on Parallel Architectures and Compilation Techniques (PACT'98)*, pages 385–392, October 1998.
- [23] J. M. Perez-Jorda and W. Yang. On the scaling of multipole methods for particle-particle interactions. *Chemical Physics Letters*, 282:71–78, 1998.
- [24] W. T. Rankin. *Efficient Parallel Implementations of Multipole Based N-body Algorithm*. PhD thesis, Duke University, 1999.
- [25] A. Snively, N. Wolter, and L. Carrington. Modeling application performance by convolving machine signatures with application profiles. In *Proceeding of the IEEE Workshop on Workload Characterization*, pages 149–156, December 2001.
- [26] B. Van de Wiele, F. Olyslager, and L. Dupre. Application of the fast multipole method for the evaluation of magneto-static fields in micromagnetic computations. *Journal of Computational Physics*, 227:9913–9932, 2008.
- [27] W. R. Wolf and S. K. Lele. Aeroacoustic integrals accelerated by fast multipole method. *AIAA Journal*, 49(7):1466–1477, 2011.
- [28] P. H. Worley. Performance evaluation of the IBM SP and the Compaq AlphaServer SC. In *Proceeding of the ACM International Conference of Supercomputing 2000*, pages 235–244, 2000.
- [29] J.-S. Zhao and W.-C. Chew. Three-dimensional multilevel fast multipole algorithm from static to electrodynamic. *Microwave and Optical Technology Letters*, 26(1):43–48, 2000.